How to Use SNVTs in LONWORKS™ Applications

August 1991 **LONWORKS Engineering Bulletin**

Introduction

Echelon's LONWORKS technology is intended for the design of distributed sense and control products. The LONTALK™ protocol, a communications protocol conforming to the seven-layer OSI Reference Model, has been optimized for intelligent distributed control applications as well as for facilitating interoperability among products using LONWORKS technology. In order to achieve interoperability, a few simple guidelines need to be followed at the physical and application layers of the protocol. Consistency at layers in-between (two through six) is achieved by using the LONTALK protocol embedded in the NEURON® CHIP.

Consistency at the application layer is facilitated through the use of Standard Network Variable Types (SNVTs). The intent of this engineering bulletin is to focus on the application layer and specifically on the proper use of SNVTs.

You will find Echelon's document *SNVTs – The Master List and Programmer's Guide* useful reference material for this engineering bulletin.

This document is divided into three sections. The first section provides an overview of network variables. The second section defines SNVTs as a class of network variables, explains why SNVTs are important, and provides guidelines on their proper use. The third section provides several lighting system examples that illustrate these guidelines and provide guidance on connecting multiple sensors and actuators. Actual NEURON C code for these examples is listed in the Appendix.

Overview of Network Variables

A network variable is a data object on one node that can be accessed by one or more additional nodes. A node's network variables define its inputs and outputs from a network point of view and allow the sharing of data in a distributed application. Whenever the application program running on a given node writes into one of its *output* network variables, the new value of the network variable is propagated across the network to all nodes with *input* network variables connected to that output network variable. This is done automatically by the LONTALK protocol. Thus, explicit messages (i.e. special instructions for sending and receiving network variable updates) updating the data across the network do not have to be generated by the application.

Network variables greatly simplify the process of developing and installing distributed systems by keeping the network configuration independent of the node's application. With network variables, nodes can be defined individually and then logically connected and reconnected to other nodes in a variety of network configurations, provided the network variable data types match. A data type is a measurement or state that is specifically defined in terms of units, range and resolution.

Standard Network Variable Types (SNVTs)

LONWORKS technology gives developers tremendous flexibility in defining network variable data types to suit their needs. However, flexibility also brings the disadvantage of independent and dissimilar definitions, limiting the chances of nodes implemented by different designers and manufacturers working together.

To facilitate interoperability among nodes, a list of approximately 47 network variables of *standard* types (SNVTs) has been defined. This list, which may grow over time, is intended to be robust enough to handle all possible LONWORKS applications yet small enough to eliminate redundancy and promote interoperability.

Developers will find that using SNVTs in conjunction with the following four guidelines will greatly enhance the chance of developing products that interoperate with other LONWORKS-based products.

Guideline #1: No Explicit Messaging

For network communications, the application should only use network variables, and not explicit messaging. Furthermore, these network variables should all have types belonging to the list of SNVTs. Echelon's tools ensure that network variables of a standard type can only be connected to network variables of the same type. This automatically prevents erroneous connections. Thus, for example, a discrete variable cannot accidentally be connected to a continuous variable.

Guideline #2: Use SNVTs to Encapsulate Hardware Details

Encapsulation of unique design details within a node (or sub-system) is a system engineering principle that simplifies design of complex systems and greatly enhances interoperability. In order to use a SNVT, data within the device must first be converted to the particular definition of the SNVT. For example, a sensor node should condition its physical inputs so that the network variables it propagates as outputs are appropriately linearized, calibrated and filtered versions of the raw measurements. Thus, a thermistor-based temperature sensor could be replaced by a thermocouple-based sensor as long as each of them generates a calibrated temperature of type *SNVT_temp*. The ability to swap one device for another facilitates interoperability. Each of these devices has a different linearity

characteristic and calibration requirement; however, this is hidden by the local processing of the sensor node.

Guideline #3: Network Variables Should Represent States, Not Commands

Network variables should always represent states and not commands. For example, an ordinary two-position light switch has two states - up and down. This should be reflected in the exported network variable from this light switch, which would be of a discrete type *SNVT_lev_disc*. The switch could be designed so that its output represented a command (turn light on, turn light off), but this would make the switch incompatible with state-oriented devices. Since all other interoperable devices will use the state interpretation of network variables, they will be able to interconnect with other devices that do the same.

Guideline #4: Logical Processing of Device State Should Occur in Actuator Node, Not Sensor Node

In general, LONWORKS devices can be classified as either sensors or actuators. Some devices incorporate elements of both. *Sensor nodes* have *input devices* (sensors) connected to their I/O pins and have *input network variables*. The input devices may be physical transducers such as for temperature or pressure, or they may be user interface inputs such as push-buttons, rotary dials and keyboards. *Actuator nodes* have *output devices* connected to their I/O pins and have *output network variables*. These output devices may be motors, lights, relays, etc.

Logical processing of device state should be carried out in the actuator nodes, not in the sensor nodes. In general, sensor-based nodes measure their inputs and then propagate the state of these inputs as output network variables. They should not make logical decisions based on the inputs. For example, a temperature sensor node should propagate the measured temperature as an output network variable, but should not decide whether to activate an air conditioning element. The decisions and control algorithms should be implemented by the actuators. Thus, an air conditioning unit would have an input network variable that would provide a measured temperature, rather than a command to turn it on. The air conditioning unit would then make its decision based on its input network variables using an appropriate algorithm.

Using SNVTs in Lighting Systems

This section focuses on a variety of lighting system examples, showing how SNVTs are used. Before describing the design of switches and lights, however, we will first discuss general design philosophy for sensors and actuators.

In a simple lighting system, the sensor devices form the user interface - devices such as switches that have discrete states, and devices such as dimmer controls that have continuous states. The actuator devices are lamps that may have discrete on/off control, and lamps that have continuous control of light intensity.

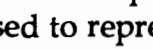
Designing Sensors (Switches) for Lighting Systems

Adhering to the guidelines, the switches should simply generate output network variables representing their states. Following are some examples of the states of a switch, and how they could be encoded using *SNVT_lev_disc*, which is a discrete SNVT, or *SNVT_lev_contin*, which is a continuous SNVT.

Variables of type *SNVT_lev_disc* can take on five enumerated values:

Enumeration Literal	Value
ST_OFF	0
ST_LOW	1
ST_MED	2
ST_HIGH	3
ST_ON	4

Variables of type *SNVT_lev_contin* can assume values in the range 0 to 200, representing a range from 0% to 100% of full scale. The resolution of this representation is in 0.5% increments.

Manufacturers can make their products more applicable by implementing several network variables in the same device. This has little impact on the cost of the device, since it is purely an enhancement to the software running on the NEURON CHIP in the node. The installer can select which network variable to connect to depending on the desired functionality. NEURON C code to implement all these examples may be found in the Appendix. In the following figures, the symbol  will be used to represent an *output* network variable, and the symbol  will be used to represent an *input* network variable.

Example 1. Two-position Toggle Switch

This example consists of one output network variable called *NV_switch*, which is of type *SNVT_lev_disc*.

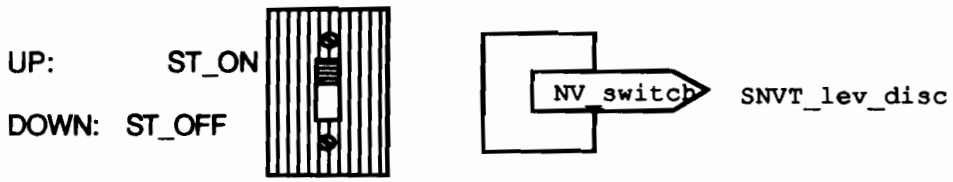


Figure 1. Two-position toggle switch.

Example 2. Push-button Switch with Two Behaviors

This example describes a push-button switch which can operate in one of two ways: (i) as a momentary switch, or (ii) as a toggle switch. The switch can be implemented with two output network variables of type *SNVT_lev_disc*. See figure 2.

(i) Momentary Push-button Switch

Network variable name: *NV_momentary*

Pushed: ST_ON

Released: ST_OFF

(ii) Toggle Push-button Switch

Network variable name: *NV_toggle*

Initially:	ST_OFF
After odd number of pushes:	ST_ON
After even number of pushes:	ST_OFF

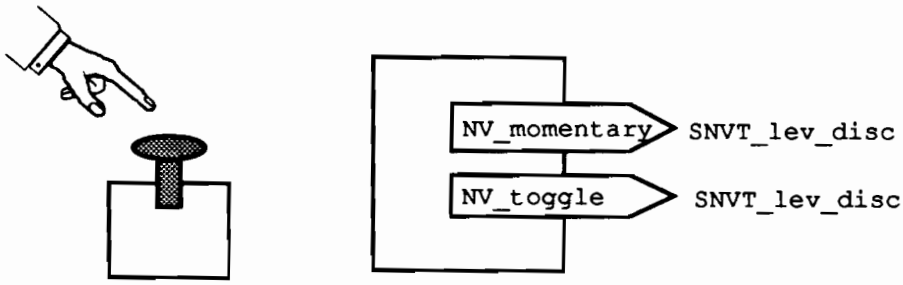


Figure 2. Push-button switch with two behaviors.

Example 3. Three-position Switch with Three Separate Behaviors

This example describes a three-position rotary or toggle switch which can be implemented with one or two output network variables of types *SNVT_lev_disc* or *SNVT_lev_contin*. See figure 3.

(i) Three-position Switch with One Output Network Variable *NV_3_disc_lev* of Type *SNVT_lev_disc*

Left position:	ST_OFF
Middle position:	ST_MED
Right position:	ST_ON

(ii) Three-position Switch with One Output Network Variable *NV_3_cont_lev* of Type *SNVT_lev_contin*

Left position:	0 %
Middle position:	50 %
Right position:	100 %

C) Three-position Switch with Two Output Network Variables *NV_device_A* and *NV_device_B*, Both of Type *SNVT_lev_disc*

This switch is used to control two devices (for example, discrete lights) - left position for device A, right position for device B. Both devices may not be on at the same time. It has the output network variables: shown in figure 3.

Switch Position	*NV_device_A*	*NV_device_B*
Left	ST_ON	ST_OFF
Middle	ST_OFF	ST_OFF
Right	ST_OFF	ST_ON

Figure 3. Three-position rotary switch.

Example 4. Three-position Switch with Two Separate Behaviors

This example describes a three-position rocker switch with a spring-loaded return to center resulting in two separate behaviors. This example uses one or two output network variables of types *SNVT_lev_disc* or *SNVT_lev_contin*.

(i) Three-position Rocker Switch with One Output Network Variable *NV_output_lev* of Type *SNVT_lev_contin*

Initially: *NV_output_lev* = 0

Push right: Increment *NV_output_lev* by some fixed amount, say 10%, for each time that it is pressed. If switch is held down for more than one second, incrementing is automatic, once per second.

Push left: Decrement *NV_output_lev* the same way.

The variable NV_output_lev is restricted to the range 0 - 100%, (numerically 0 - 200), since it is of type *SNVT_lev_contin*.

(ii) Three-position Rocker Switch with Two Output Network Variables *NV_device_A* and *NV_device_B*, Both of Type *SNVT_lev_disc*

This example shows the control of two devices.

Initially: Both devices are off

Push left: Toggle the state of device A between ST_OFF and ST_ON

Push right: Toggle the state of device B between ST_OFF and ST_ON

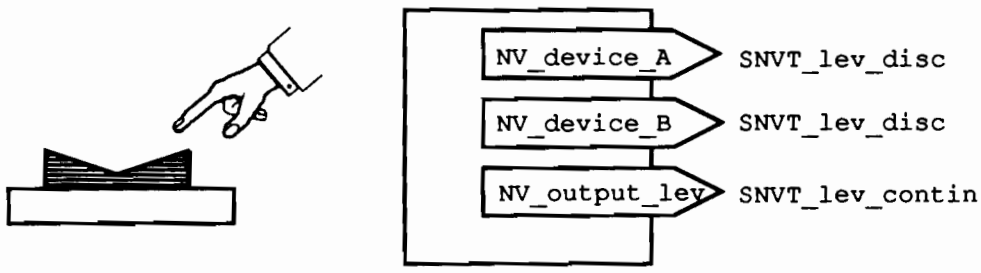


Figure 4. Three-position spring-loaded rocker switch.

There are many other possibilities of user interface devices using switches. The examples above are those familiar to most users for a number of different devices. For example, many home appliance remote control devices use a three-position switch (or two momentary push-buttons) to control a continuous output such as a volume control.

Devices which have multiple output possibilities are installed by connecting to the appropriate network variables and perhaps installing a face plate with a legend indicating which functions of the switch are being used. This extra flexibility allows the same device to be used in different scenarios without the expense of stocking multiple types of the control devices.

Continuous control devices (such as dimmer controls) are devices that output a continuous level which can be represented by a value of the type *SNVT_lev_contin*. This data type represents a value in the range 0% to 100%, with a resolution of 0.5%. It is an eight-bit unsigned quantity in the range of 0 to 200. The control could be a shaft encoder - a device that generates a pair of pulse trains in quadrature with each other. The NEURON CHIP can decode the pulse trains from such a device to determine the amount and direction of the rotation as the user turns the knob. See Echelon's application note titled *NEURON® CHIP Quadrature Input Function Interface* for more details on quadrature devices. Other possibilities include an analog potentiometer with an analog-to-digital conversion circuit to measure the voltage, and hence the rotational angle.

Designing Lighting System Actuators (Lights)

There are many considerations in the design of a light fixture which are beyond the scope of this engineering bulletin. Lights may be incandescent or fluorescent, AC or DC, low or high voltage, etc. We will assume that there is only one characteristic of a light that is important here - whether its light intensity is controlled discretely (i.e. it can only be turned on or off), or continuously (i.e. a triac-controlled AC incandescent, a dimmable fluorescent, or a pulse-width modulated DC lamp).

The simplest discrete (on/off) light may be modeled as a node that has one input network variable of type *SNVT_lev_disc*. How should it behave? If the input network variable has the value ST_OFF, then the light should go off. If the input has the value ST_ON, then the light should go on. But if the input has one of the values ST_LOW, ST_MED or ST_HIGH, then we shall establish a convention that the light will go on. Therefore, this light can be controlled by any switch that has an output network variable of type *SNVT_lev_disc*.

But what happens if we want to control the light with multiple discrete-type switches? There are several possibilities - the light designer can choose the behavior he desires. If we simply connect all the output network variables from the switches to the input of the light, as in figure 5, then the behavior is such that the last switch to be toggled will control the state of the light. So if the light is off, and the user wishes to turn it on, he may simply move any switch to the up position. But if the

switch is already in the up position, he may have to move it to the down position (no effect on the light) and then move it back to the up position to turn the light on. Similarly, he may have to toggle the switch up and down to turn the light off. Note that just looking at the switch does not tell one whether the light is on - this will always be true when there are multiple switches controlling a single light.

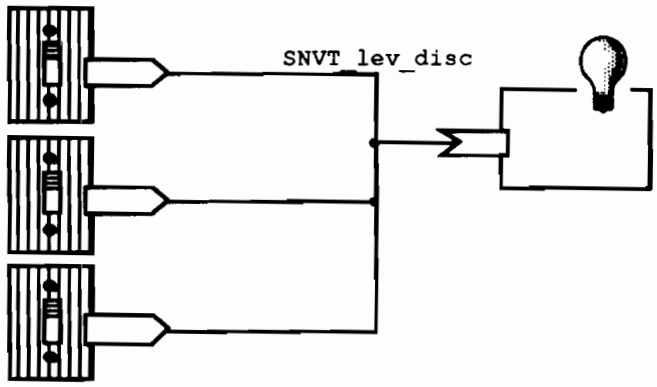


Figure 5. Multiple switches controlling a single light.

An alternative application allows one to change the state of the light by moving the switch only once. Thus, if the light is off, but the switch is up, then moving it down will turn the light on. This kind of behavior is common in two-way wiring installations using conventional technology. But with LONWORKS technology, we can extend this functionality to three or more switches. In figure 6, we show a light that allows up to eight switches. The light requires a separate input network variable for each switch that is to be connected. This allows the light to detect a change in each individual switch. One to eight of these network variables may be connected to discrete-type switches at the time the light is installed.

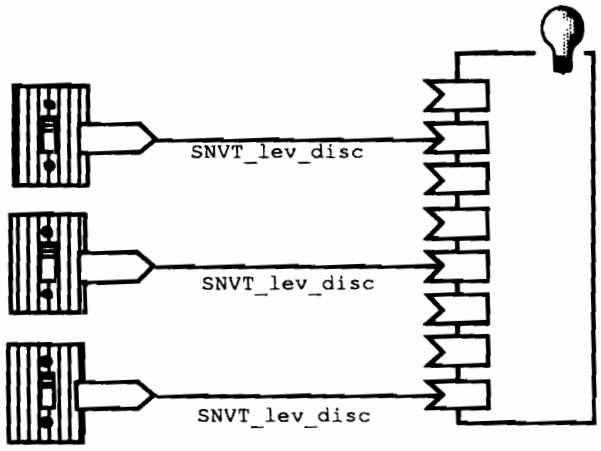


Figure 6. Multiple switches controlling a single light (improved).

The logic in the light simply counts the number of switches that are on. If there are an odd number, then the light goes on, if there are an even number, then the light goes off. Since moving a switch will always change the parity of the number of switches that are turned on, the light will always change state when a switch is moved. NEURON C code for this example is presented in the Appendix.

A discrete light may be made more flexible by allowing it to be controlled by a continuous type control (see figure 7), as well as a discrete control. For example, dimmer dials can be used to turn a light on and off if necessary. The light makes a policy decision. For example, if the continuous level is greater than 50%, it goes on; if it is less than 50%, it goes off. This decision is arbitrary, since the threshold level may be set anywhere. There could also be a configuration network variable that sets the threshold level. The installer determines the desired level by sending a one-time network variable update to the node when it is installed. For a code example, see the Appendix.

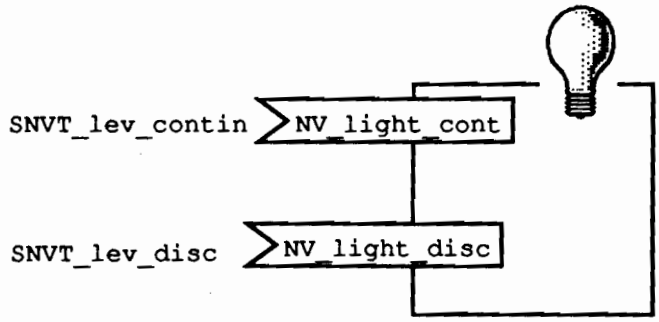


Figure 7. Light with discrete and continuous controls.

Continuous-type (dimmable) lights can obviously have an input variable of type *SNVT_lev_contin*. But they can also have an input variable of type *SNVT_lev_disc*, and make the decision, say, to control the light intensity according to some formula. For example, ST_OFF -> 0%, ST_LOW -> 25%, ST_MED -> 50%, ST_HIGH -> 75% and ST_ON -> 100%. For a code example, see the Appendix. Figure 7 also applies to this example.

What happens if we want to control a continuous (dimmable) light from multiple continuous-type dimmer dials? The obvious solution is to connect all the control outputs to the input of the light as shown in figure 8.

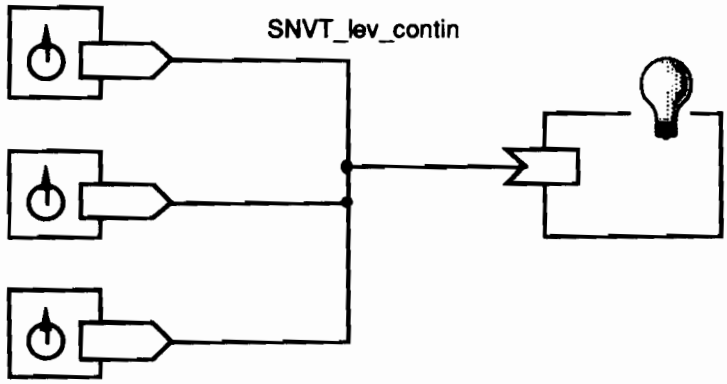


Figure 8. Multiple dimmers controlling a single light.

The behavior of this arrangement is that the last dimmer control to be turned sets the intensity of the light. But that means that if one dimmer control has been used to turn the light down, and then a second dimmer control is used to turn the light up, then touching the first dimmer control will cause the brightness to jump discontinuously back to a low intensity. This may not be desirable.

A better behavior would be if the dimmer control changes the lamp intensity relative to its current value. This can be achieved by feeding the current lamp brightness back to the dimmer controls, so that they can output a desired brightness level based on the current brightness level of the lamp. This considerably complicates the design.

Appendix

This Appendix contains NEURON C code showing the use of SNVTs in the switch examples mentioned above. In all of these examples, it is assumed that the switches are normally open, connected between the NEURON CHIP's I/O pins and ground. The on-chip pull-up resistors on pins IO.4 through IO.7 are used so that the input pins are logic one when the switches are open, and logic zero when closed. These resistors are enabled (see figure 9) by the statement:

#pragma enable_io_pullups

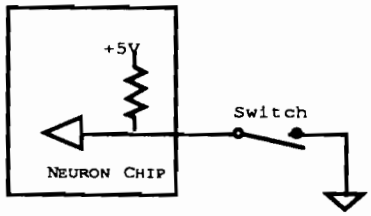


Figure 9. Using the NEURON CHIP's pull-up resistors.

```
// Two position toggle switch - when UP, grounds pin IO_4 (Fig. 1)

#pragma enable_io_pullups
#include <SNVT_lev.h>
IO_4 input bit IO_toggle;
network output SNVT_lev_disc NV_switch = ST_OFF;   // when UP, set to ST_ON

when( io_changes( IO_toggle ) ) {
    if( input_value == 0 ) NV_switch = ST_ON;
       else NV_switch = ST_OFF;
}
```

```
// Push-button switch - when pressed, grounds pin IO_4
// Has two output network variables - one momentary, one toggle (Fig. 2)

#pragma enable_io_pullups
#include <SNVT_lev.h>
IO_4 input bit IO_button;

network output SNVT_lev_disc NV_momentary = ST_OFF;    // when PRESSED, set to ST_ON
network output SNVT_lev_disc NV_toggle = ST_OFF;       // alternates states

when( io_changes( IO_button ) ) {
    if( input_value == 0 ) NV_momentary = ST_ON;
       else NV_momentary = ST_OFF;
    if( NV_toggle == ST_OFF ) NV_toggle = ST_ON;
```

```
        else NV_toggle = ST_OFF;
}
```

```
// Three position rotary switch  - Left position grounds pin IO_4,
// Middle position grounds pin IO_5 - Right position grounds pin IO_6
// Has three different behaviors (Fig. 4)

#pragma enable_io_pullups
#include <SNVT_lev.h>

IO_4 input bit IO_left;
IO_5 input bit IO_middle;

#pragma enable_io_pullups
#include <SNVT_lev.h>

IO_4 input bit IO_left;
IO_5 input bit IO_middle;
IO_6 input bit IO_right;

network output SNVT_lev_disc NV_device_A = ST_OFF;        // when  left pushed, set to ST_ON
network output SNVT_lev_disc NV_device_B = ST_OFF;        // when right pushed, set to ST_ON

network output SNVT_lev_disc NV_3_disc_lev = ST_OFF;      // ST_OFF, ST_MED or ST_ON
//      Values of SNVT_lev_disc:
//      ST_OFF = 0, ST_LOW = 1, ST_MED = 2, ST_HIGH = 3, ST_ON = 4;

network output SNVT_lev_contin NV_3_cont_lev = 0;         // in 0.5 % steps

when( io_changes( IO_left ) to 0 ) {
    NV_device_A = ST_ON;
    NV_device_B = ST_OFF;
    NV_3_disc_lev = ST_OFF;
    NV_3_cont_lev = 0;
}

when( io_changes( IO_middle ) to 0 ) {
    NV_device_A = ST_OFF;
    NV_device_B = ST_OFF;
    NV_3_disc_lev = ST_MED;
    NV_3_cont_lev = 100;  // 50 %
}

when( io_changes( IO_right ) to 0 ) {
    NV_device_A = ST_OFF;
    NV_device_B = ST_ON;
    NV_3_disc_lev = ST_ON;
    NV_3_cont_lev = 200;  // 100 %
}
```

```
// Three position rocker switch with spring-loaded center
// When left side pressed, grounds pin IO_4
// When right side pressed, grounds pin IO_5
// Has two different behaviors (Fig. 4)

#pragma enable_io_pullups
#include <SNVT_lev.h>
IO_4 input bit IO_left;
IO_5 input bit IO_right;

network output SNVT_lev_disc NV_device_A = ST_OFF;
network output SNVT_lev_disc NV_device_B = ST_OFF;
network output SNVT_lev_contin NV_output_lev = 0; // in 0.5 % steps

mtimer repeating increment_timer;       // timers to control auto-repeat
mtimer repeating decrement_timer;

void increment_NV( void ) {      // function to increment NV_output_lev
    if( NV_output_lev < 200 ) NV_output_lev = NV_output_lev + 20;    // add 10 %
}
void decrement_NV( void ) {      // function to decrement NV_output_lev
    if( NV_output_lev > 0 ) NV_output_lev = NV_output_lev - 20;  // subtract 10%
}

when( io_changes( IO_left ) to 0 ) {
    if( NV_device_A == ST_OFF ) NV_device_A = ST_ON;      // toggle device A
       else NV_device_A = ST_OFF;
    decrement_NV( );
    decrement_timer = 1000;      // set timer for once a second
}
when( io_changes( IO_right ) to 0 ) {
    if( NV_device_B == ST_OFF ) NV_device_B = ST_ON;      // toggle device B
       else NV_device_B = ST_OFF;
    increment_NV( );
    increment_timer = 1000;      // set timer for once a second
}
when( io_changes( IO_left ) to 1 ) {       // left switch is released
    decrement_timer = 0;                   // stop timer
}
when( io_changes( IO_right ) to 1 ) {      // right switch is released
    increment_timer = 0;                   // stop timer
}
when( timer_expires( decrement_timer ) {
    decrement_NV( );
}
when( timer_expires( increment_timer ) {
    increment_NV( );
}
```

NEURON C code for the light examples - light control function prototypes are defined, the implementation depends on the particular hardware.

```
// Discrete light with up to eight discrete inputs (Fig. 6)

void turn_light_on( void );      // hardware-dependent functions
void turn_light_off( void );        // not supplied here
#include <SNVT_lev.h>

network input SNVT_lev_disc NV_light_control[ 8 ]; // Uses network variable array for clarity

when( nv_update_occurs ) {       // any time any input changes
    boolean light_is_on;
    int i;

    light_is_on = FALSE;      // compute parity of the inputs (even or odd)
    for( i = 0; i < 8; i++ )
        if( NV_light_control[ i ] != ST_OFF ) light_is_on = !light_is_on;

    if( light_is_on ) turn_light_on( );
        else turn_light_off( );
}
```

```
// Discrete light with continuous input, variable threshold, and discrete input (Fig. 7)

void turn_light_on( void );      // hardware-dependent functions
void turn_light_off( void );        // not supplied here
#include <SNVT_lev.h>

network input SNVT_lev_contin NV_light_cont;   // continuous input
network input SNVT_lev_disc NV_light_disc; // discrete input

config network input SNVT_lev_contin NV_threshold = 100;
       // threshold level - default = 50 %

when( nv_update_occurs( NV_light_cont ) ) {
    if( NV_light_cont > NV_threshold ) turn_light_on( );
        else turn_light_off( );
               // turn light on if continuous input exceeds threshold
}

when( nv_update_occurs( NV_light_disc ) ) {
       // turn light off if discrete input is ST_OFF
       // turn light on if discrete input is ST_LOW, ST_MED, ST_HIGH or ST_ON

    if( NV_light_disc != ST_OFF ) turn_light_on( );
        else turn_light_off( );
```

```
// Continuously dimmable light with continuous and discrete inputs (Fig. 7)

#include <SNVT_lev.h>

void set_light_intensity( SNVT_lev_contin );
// hardware-dependent function, not supplied here

network input SNVT_lev_contin NV_light_cont;   // continuous input
network input SNVT_lev_disc NV_light_disc; // discrete input

when( nv_update_occurs( NV_light_disc ) ) {        // handle discrete input
    SNVT_lev_contin brightness;

    switch( NV_light_disc ) {
        case ST_OFF:   brightness =   0; break; //   0%
        case ST_LOW:   brightness =  50; break; //  25%
        case ST_MED:   brightness = 100; break; //  50%
        case ST_HIGH:  brightness = 150; break; //  75%
        case ST_ON:    brightness = 200; break; // 100%
    }
    set_light_intensity( brightness );      // update lamp hardware
}

when( nv_update_occurs( NV_light_cont ) ) {        // handle continuous input
    set_light_intensity( NV_light_cont );          // update lamp hardware
}
```

Disclaimer

Echelon Corporation assumes no responsibility for any errors contained herein.
No part of this document may be reproduced, translated, or transmitted in any form without permission from Echelon.

© 1991 Echelon Corporation. ECHELON, LON, and NEURON are U.S. registered trademarks of Echelon Corporation. LONMANAGER, LONBUILDER, LONTALK, LONWORKS, 3150, and 3120 are trademarks of Echelon Corporation. Patented products. Other names may be trademarks of their respective companies. Some of the LONWORKS TOOLS are subject to certain Terms and Conditions. For a complete explanation of these Terms and Conditions, please call 1-800-258-4LON.

Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
105 Heath Street
London NW3 6SS
England
Telephone (071) 431-1600
Fax (071) 794-0532
International Telephone + 44 71 431-1600
International Fax + 44 71 794-0532

Echelon Japan K.K.
AIOS Gotanda Building #808
10-7, Higashi-Gotanda 1-chome,
Shinagawa-ku, Tokyo 141, Japan
Telephone (03) 3440-8638
Fax (03) 3440-8639

Part Number 005-0002-01